CSYE7374 Final Report

(Gender_Detection_Face)

Prepared by

Dingling Ge, Zefan Feng

(04/11/2020)

TABLE OF CONTENTS (TOC)

1.0 Introduction

- 1.1 Background of this project
- 1.2 Objectives

2.0 Methodology

- 2.1 Problem Analysis
- 2.2 Algorithms and software libraries
  - 2.2.1 Why Artificial Neural network (ANN)
  - 2.2.2 Why Convolutional Neural Network (CNN)
  - 2.2.3 Why Tensorflow

2.2.4 Why Keras

2.2.5 Why OpenCV

2.2.6 Why Parallelized (multi-CPU) by OpenMP (Wrapped by Tensorflow)

3.0 Specifications of the dataset

3.1 Data Description

3.2 Data Resource

4.0 Main Realization Process

4.1 Load Data

4.2 Shuffle Data

4.3 Construct ANN model and training

4.4 Predict the gender and Evaluate the results by ANN

4.5 Construct CNN model and training

4.6 Predict the gender and Evaluate the results by CNN

4.7 Parallelism

5.0 Experiment, Results and Analysis

5.1 Performance of different sizes

5.1.1 Try 28*28

5.1.1.1 ANN vs CNN of training performance

5.1.1.2 ANN vs CNN of test predict performance

5.1.2 Try 100*100

5.1.2.1 ANN vs CNN of training performance

5.1.2.2 ANN vs CNN of test predict performance

5.2 Performance of different Threads based on DNN model

5.2.1 Change threads based on 28*28

5.2.2 Change threads based on 100*100

6.0 Conclusion

7.0 Reference

8.0 Deliverables

1.0 Introduction

This is the final report of Group12 on Gender_Detection_Face for CSYE7374 Parallel Machine Learning&AI Spring2020.

1.1 Background of this project

Human face contains rich information, such as gender, age, race, expression, etc., and the application of facial image has attracted wide attention. Among them, gender is the most basic information you need to know to classify humans. Generally, humans can intuitively and easily judge a person ’s gender by using human eyes, but it is not easy for a computer to achieve the same function. In recent years, with the continuous in-depth development of computer science and technology, the functions of computers have become more and more rich and powerful, and more and more intelligent. The function of imitating various biological features of humans has been a research hotspot in the computer world. As an important research direction of intelligent human-computer interaction technology, face gender recognition technology has received extensive attention and rapid development in recent years. It has important theoretical significance and application prospects in identity authentication, video retrieval and robot vision. In this project, we are trying to approach a precise binary classification between biological males and females.

The gender detection algorithm is based on features of the face. It detects and maps out a series of points on the face, such as corners and edges of the eyes, lips and nostrils. By running those points through algorithms, it can extract information of interest from those points and concatenate different pieces of information into a single feature. The gender of a person can, therefore, be detected based on how the features tend to appear for a certain gender. In this project, we mainly use convolutional neural networks to achieve gender detection, and in terms of performance, consider optimizing it in parallel, and do various experiments to support it.

1.2 Objectives

This project supports the following objectives:

- Choose an algorithm to implement gender detection (CNN and ANN?)
- Optimize performance through parallelism (multi-CPU)

- Experimental support
- Analysis
- Make a conclusion

2.0 Methodology

2.1 Problem Analysis

Image classification problem, which is the task of assigning a label to the input image from a fixed set of classifications. This is one of the core problems of computer vision. In this project, the image classification model requires a single image and assigns the probability to 2 labels {man, woman}. For computers, the image is represented as a large 3-dimensional array of numbers. For example, take a picture of this project (from dataset/test/man / face_1.jpg like below), the image is 106 pixels wide and 146 pixels high, and has three color channels red, green, and blue (or RGB for short). Therefore, the image consists of 106 x 146 x 3 digits, for a total of 46428 digits. Each number is an integer, ranging from 0 (black) to 255 (white). Our task is to turn this quarter number into a single label, such as "man". For the convenience of calculation, we can convert the white RGB three-channel image into a grayscale image for processing, and finally predict the classification to which the picture belongs, and compare it with the original labeled label to see the performance. So, in general, this is a binary classification problem.

face_01.jpg

It can be seen from the above analysis that because the image features are each of its pixels, the color picture features are even three times the grayscale image features, so compared to the CSV dataset, the features of the picture file are far Much larger than the CSV dataset. At the same time, the size of the picture will also affect the number of features. The larger the picture size, the more features will increase. Therefore, the image data set usually takes more time to train. But we want to increase the speed of training and get results in less time, so the parallel is our primary choice to improve performance.

2.2 Algorithms and software libraries

2.2.1. Why Artificial Neural network (ANN)

Traditional machine learning algorithms such as logistic regression and random forest usually perform poorly in classification tasks due to the variety and complexity of the features of images, so we do not consider using them.

We first consider the neural network algorithm among the existing algorithms. The artificial neural network algorithm is the basis of the neural network algorithm, the greatest power is that they can think like human--solve the same type of problem once familiar with a problem in the training process.

The smallest and most important unit in an artificial neural network is called a neuron. Just like the biological nervous system, these neurons are interconnected and have powerful processing power. In general, ANN tries to display the behavior and processes that happen in real brains, and that's why their structures are modeled based on biological observations.

The same goes for artificial neurons, whose structure is reminiscent of real neurons. Each neuron has an input connection and an output connection. These connections mimic the behavior of synapses in the brain. In the same way that synapses in the brain transmit signals -- from one neuron to another -- these connections also carry information between artificial neurons. Each connection has a weight, which means that the value sent to each connection is multiplied by this factor. This pattern is inspired by synapses in the brain, and the weights actually simulate the amount of

neurotransmitters that pass between biological neurons. That is, if a connection is important, it will have more weight than if it is not important.

Many values may enter into a neuron, so each neuron has a so-called input function. Generally, the continuous input values are weighted and summed. This value is then passed to the activation function, which calculates whether or not this signal is sent to the neuron's output. We can also have multiple layers of neurons in each ANN.

Overall, first we randomly initialize the weights in the neural network, and send the first set of input values to the neural network, then transmit them through the network and get the output value. Next compare the output value with the expected output value and calculate the error using the cost function. The error is propagated back to the network and the weight is set accordingly. Repeat the steps above after initializing the weight for each input value in the training set. Once the entire training set was sent to the neural network, we completed an epoch and repeated epochs several times.

This is a simplified representation of how artificial neural networks learn. In practice, the training set is divided into two parts, and the second part is used to verify the network.

Therefore, using artificial neural networks(ANN) to classify images is a worthwhile attempt.

2.2.2. Why Convolutional Neural Network (CNN)

We first tried using ANN to classify images. Although it performs better than traditional machine learning methods, it still can be improved. Then, we would use one Neural Network structure, Convolutional Neural Network (CNN), which is more suitable for image classification tasks. In recent years, almost all the major breakthroughs in image recognition area have been made by convolutional neural networks(CNN),

Artificial neural network neural network is not as good as CNN, mainly due to the following problems:

- Too many parameters. For a large size image, such as 200x200x3, it would return neurons with a weight of 2002003 = 120,000. This full connection is a waste, and the large number of parameters can quickly lead to over-matching.
- For an image recognition task, each pixel is closely related to its surrounding pixels, while the connection with pixels far away might be small. If a neuron is

connected to all the neurons in the upper layer, which means treating all the pixels of the image the same for one pixel, and that is not consistent with the previous assumption. After we learned the weights of each connection, we may end up with a large number of weights whose values are small (that is, the connections don't really matter). The effort to learn a lot of unimportant weights is bound to be very inefficient.

- Limits of network layers. We know that the more layers the network has, the more expressive it is. However, it is difficult to train deep artificial neural networks by gradient descent method, because the gradient of a fully connected neural network is difficult to transmit more than 3 layers. Therefore, we cannot get a deep fully connected neural network, which also limits its performance.

Convolution neural network(CNN) can solve this problem, however, there are three main ideas:

- Local connections: each neuron is no longer connected to all the neurons in the upper layer, but to a small number of neurons, which reduces a lot of parameters.
- Weight sharing: a set of connections can share the same weight instead of each connection having a different weight, which also reduces the number of parameters.
- Downsampling: Pooling can be used to reduce the number of samples in each layer to further reduce the number of parameters, and improve the robustness of the model. For the image recognition task, the convolutional neural network(CNN) achieves better learning effect by maintaining as many important parameters as possible and removing a large number of unimportant parameters.

The convolutional neural network（CNN） consists of one or more convolutional layers, pooling layers and fully connected layers. Compared with other deep learning structures, convolutional neural networks(CNN) can give better results in image and other aspects. This model can also be trained using a back propagation algorithm. Compared with other shallow or deep neural networks, convolutional neural networks(CNN) have fewer parameters to consider, making it an attractive deep learning structure.

2.2.3. Why Tensorflow

Tensorflow as one of the most popular libraries for deep learning, is a lightweight software that can instantly generate your training model and re-implement it, running on all types of machines from supercomputers to embedded systems. Its distributed architecture enables model training for large data sets to not require much time. TensorFlow can provide all of our frameworks and API, and can run on multiple CPU, GPU or both at the same time, which is convenient for parallel implementation. Therefore, tensorflow is our first choice.

2.2.4. Why Keras

Keras is just a superstructure of deep learning modeling, with the flexible use of TensorFlow at the back end. In this way, we can avoid the differences between different deep learning frameworks and focus on the modeling process. And you can seamlessly switch between CPU and GPU. Just as jQuery is a layer of encapsulation for JavaScript, Keras is a layer of encapsulation for machine learning frameworks like CNTK, TensorFlow, or Theano, greatly simplifying the load of code. Having advantages like simple API, easy to use, complete documentation, good scalability.

2.2.5 Why OpenCV

OpenCV is a powerful cross-platform computer vision library distributed based on the BSD license (open source). For image processing, the image opened by OpenCV can truly and losslessly display the information of the image, which is at the pixel level and it can more accurately process the image with strong scalability and high customization. Therefore we chose OpenCV to do the basic operation for image processing.

2.2.6 Why Parallelized (multi-CPU) by OpenMP (Wrapped by Tensorflow)

CPU parallel processing is actually what we call multithreading. Serial processing is like putting two tasks or several tasks together and the CPU needs to complete one by one

because the CPU has only one way to go. While the parallel processing source is like dividing the task into multiple ones and supporting a CPU to process in the divided two or more ways, which is equal to the completion of multiple tasks in one time, which is obviously more efficient.

Parallelism can significantly improve the performance. In this project, we chose to implement parallelism with OpenMP, which is encapsulated in tensorflow and can be used directly. We will give more detailed explanations later in this report.

3.0 Specifications of the dataset

3.1 Data Description

This data set contains 2280 cropped faces (1140 male and 1140 female), and it is the data set Kaggle labeled with face gender recognition in 2018, which collected from the color face images of celebrities from the Internet in Europe and U.S , which includes the training set (female/male, 800), the validation set (female/male, 170), the test set (female/male 170), and the data set have different factors such as illumination, wearing glasses, the head part. Also the size of each picture is different, each picture takes up less space since it is cropped.

Each image in the data set is marked with a label as the target value, and each pixel in each channel can be a feature.

3.2 Data Resources

Link：https://www.kaggle.com/gmlmrinalini/genderdetectionface

4.0 Main Realization Process

4.1 Load data

Using opencv to load data for testing from ./dataset/test/, and returns np arrays, then convert the original image to grey scaled image and resize to 28*28. The same as the training dataset.

```
# Load data for testing from ./dataset/test/
# Returns np arrays
def load_test():
    tran_imags = []
    labels = []
    seq_names = ['man', 'woman']
    for seq_name in seq_names:
        frames = sorted(os.listdir(os.path.join(
            './', 'dataset', 'test', seq_name)))
        for frame in frames:
            imgs = [os.path.join('./', 'dataset', 'test', seq_name, frame)]
            img = cv2.imread(imgs[0])

            # Convert to grey scaled image and resize to 28*28
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img = cv2.resize(img, (IMAGE_SIZE,IMAGE_SIZE), interpolation = cv2.INTER_AREA)
            imgs = np.array(img)
            tran_imags.append(imgs)
            if seq_name == 'man':
                labels.append(0)
            else:
                labels.append(1)
    return np.array(tran_imags), np.array(labels)
```

4.2 Shuffle data

Shuffle data every time to get different train/validate dataset, because the assumptions and data requirements of machine learning are to satisfy independent and identical distribution. It can prevent model jitter during training, which is conducive to the robustness of the model, and prevent overfitting and make the model learn more correct features, and make the data distribution of training set, validation set and test set similar.

```
        for i in range(1,len(train_data)):
            index.append(i)
        shuffle(index)
        shuffled_train_data =[]
        shuffled_train_label =[]
        for i,v in enumerate(index):
            shuffled_train_data.append(train_data[v])
            shuffled_train_label.append(train_label[v])
        shuffled_train_data = np.array(shuffled_train_data)
        shuffled_train_label = np.array(shuffled_train_label)
```

4.3 Construct ANN model and training

We first initialise the ANN, adding the input layer and the first hidden layer, and adding the second layer and output layer. Then compile the ANN model. We specify that the activation function of the output layer is softmax (this is normalized multi-classification, and the K-dimensional real number field can be compressed into the (0, 1) value range, and the sum of K values is 1), and the rest is rule . Specify the optimizer as ‘adam’ and the loss function as the cross-entropy loss.

```
model = Sequential()
model.add(Flatten())
model.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_shape=(28, 28, 1)))
model.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units = 2, kernel_initializer = 'uniform', activation = 'softmax'))
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Then, we fit the ANN to the training set, train with 15 epochs, which leads to a nearly converged result. The validation rate is set to 20% of the total training dataset.

```
history = model.fit(shuffled_train_data, keras.utils.to_categorical(
    shuffled_train_label), batch_size=32, epochs=15, verbose=1, shuffle=True,validation_split=0.2)
```

4.4 Predict the gender and Evaluate the results by ANN

Define a manOrWoman function to better express the results. For evaluation, we use cross entropy and accuracy to see training performance, and Ground truth and False positive to see the predicted performance. And the Specific Analysis of the model and Results will be explained in the 6.0 part of the report.

```
def manOrWoman(predictedResult, index):
    if predictedResult[index][0]>predictedResult[index][1]:
        return 'man'
    else:
        return 'woman'
```

```
# Predict the test dataset and summary the value
result = model.predict(test_data)
resultStats = [0,0,0,0]

for i,r in enumerate(result):
    print('truth: man' if test_label[i]==0 else 'truth: woman')
    if r[0]>r[1]:
        print('predicted: man')
    else:
        print('predicted: woman')
```

```
for i,r in enumerate(result):
    if(test_label[i]==0):
        if r[0]>r[1]:
            resultStats[0] +=1
        else:
            resultStats[1] +=1
    else:
        if r[0]>r[1]:
            resultStats[2] +=1
        else:
            resultStats[3] +=1

print("test dataset prediction result:")
print("Man :")
print("\tGround truth: ", resultStats[0]/(resultStats[0]+resultStats[1]))
print("\tFalse positive: ", resultStats[2]/(resultStats[2]+resultStats[3]))

print("Woman :")
print("\tGround truth: ", resultStats[3]/(resultStats[2]+resultStats[3]))
print("\tFalse positive: ", resultStats[1]/(resultStats[0]+resultStats[1]))
```

4.5 Construct CNN model and training

Based on the CNN structure definition, We try to construct a multi convolutional layer, and define the size (odd number), step size, etc. of the filter (with weights in it), doing the linear regression calculation.

The main function of the pooling layer is downsampling. By removing unimportant samples in the function diagram, the number of parameters is further reduced. Here we use the Max Pooling method. And to reduce overfitting, we need to add a dropout layer. Repeat the operation to get the structure which we want to construct.

Last, we need FC layer, the previous convolution and pooling operations are equivalent to feature engineering, and the full connection behind is equivalent to feature weighting. The last fully connected layer acts as a "classifier" in the entire convolutional neural network.

To get more detail of our CNN model :

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer1_con1 (Conv2D)         (None, 28, 28, 32)        320
_________________________________________________________________
layer1_con2 (Conv2D)         (None, 28, 28, 32)        9248
_________________________________________________________________
layer1_pool (MaxPooling2D)   (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
layer2_con1 (Conv2D)         (None, 14, 14, 64)        18496
_________________________________________________________________
layer2_con2 (Conv2D)         (None, 14, 14, 64)        36928
_________________________________________________________________
layer2_pool (MaxPooling2D)   (None, 7, 7, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               401536
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
=================================================================
Total params: 466,786
Trainable params: 466,786
Non-trainable params: 0
_________________________________________________________________
```

```
# Construct CNN model
model = Sequential()
# Convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', data_format='channels_last',name='layer1_con1',input_shape=(55, 55, 1)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', data_format='channels_last',name='layer1_con2'))
# Pooling layer
model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2), padding = 'same', data_format='channels_last',name = 'layer1_pool'))
# Dropout layer
model.add(Dropout(0.25))
# Another coinvolutional layer
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', data_format='channels_last',name='layer2_con1'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', data_format='channels_last',name='layer2_con2'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=(2,2), padding = 'same', data_format='channels_last',name = 'layer2_pool'))
# Flatten the matrix
model.add(Flatten())
# Dense layer
model.add(Dense(128, activation='relu'))
# Another dropout layer
model.add(Dropout(0.5))
# Classify into 2 classes.
model.add(Dense(2, activation='softmax'))
model.summary()
# Define parameters
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])

# Normalize input
shuffled_train_data = shuffled_train_data.reshape(shuffled_train_data.shape[0],55,55,1)/255
```

4.6 Predict the gender and Evaluate the results by CNN

Coding is the same as the ANN process, we will give the details and results for the specific analysis of the model in the 6.0 part of the report.

4.7 Parallelism

In Tensorflow /compiler /xla /service /backend.h defines that:

```
Backend(se::Platform* platform, Compiler* compiler,
          absl::Span<se::StreamExecutor* const> stream_executors,
          TransferManager* transfer_manager,
          ComputationPlacer* computation_placer,
          int intra_op_parallelism_threads);
```

Further we investigate in the implementation:

```
Backend::Backend(se::Platform* platform, Compiler* compiler,
                 absl::Span<se::StreamExecutor* const> stream_executors,
                 TransferManager* transfer_manager,
                 ComputationPlacer* computation_placer,
                 int intra_op_parallelism_threads)
    : platform_(platform),
      compiler_(compiler),
      transfer_manager_(transfer_manager),
      computation_placer_(computation_placer),
      stream_executors_(stream_executors.begin(), stream_executors.end())
    {
    // Create a memory allocator for the valid stream executors.
    memory_allocator_ = absl::make_unique<se::StreamExecutorMemoryAllocator>(
      platform, stream_executors_);
    CHECK(!stream_executors_.empty())
      << "Service found no devices for backend " << platform_->Name() << '.';

    if (platform->id() == se::host::kHostPlatformId) {
    const int num_threads = intra_op_parallelism_threads > 0
                                ? intra_op_parallelism_threads
                                : tensorflow::port::MaxParallelism();
    intra_op_thread_pool_.reset(new IntraOpThreadPool(num_threads));
    }
}
```

Notice that the parameter intra_op_parallelism_threads redefines the size of the IntraOpThreadPool, which is a structure consisting of a Tensorflow thread pool and a pointer of eigen thread pool devices.

```
1.  void DeviceBase::set_eigen_cpu_device(Eigen::ThreadPoolDevice* d) {
2.    // Eigen::ThreadPoolDevice is a very cheap struct (two pointers and
3.    // an int).  Therefore, we can afford a pre-allocated array of
4.    // Eigen::ThreadPoolDevice.  Here, we ensure that
5.    // Eigen::ThreadPoolDevices in eigen_cpu_devices_ has increasingly
6.    // larger numThreads.
7.    for (int i = 1; i <= d->numThreads(); ++i) {
8.      eigen_cpu_devices_.push_back(new Eigen::ThreadPoolDevice(
9.          d->getPool(), i /* numThreads() */, d->allocator()));
10.   }
11. }
```

Through the code we can see that IntraOpThreadPool actually defines the quantity of CPU eigen computation devices. It then works as a member of OpKernel to participate in the calculation.

So, first control how many threads Tensorflow uses, and util function to print time, then Initial tensorflow session with indicated threads count, and set start time and end time to calculate the run time. Change the threads, we can see the time changes by using multi CPU.

```
# Control how many threads tensorflow uses.
THREAD_NUM = 1
```

```
# Util function to print time.
def printTime():
    tmpTime = localtime()
    print(strftime("%Y-%m-%d %H:%M:%S", tmpTime))
    return tmpTime
```

```
    print('\n\nStart training with %d threads.\n\n' % THREAD_NUM)
    # Initial tensorflow session with indicated threads count.
    with tf.Session(config=tf.ConfigProto( inter_op_parallelism_threads=1, intra_op_parallelism_threads=THREAD_NUM)) as sess:
        K.set_session(sess)
```

5.0 Experiments, Results and Analysis

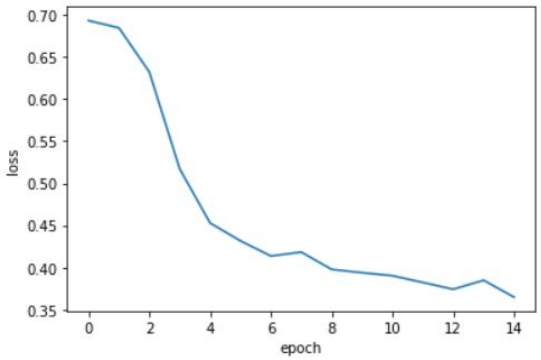
5.1 Performance of different sizes

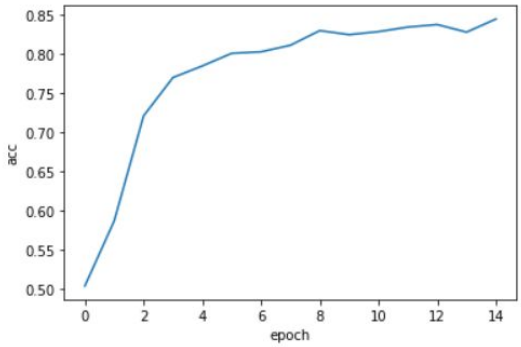
5.1.1 Try 28*28

5.1.1.1 ANN vs CNN of training performance:

For ANN, after 15 echos, it can achieve 0.8446 of the accuracy and 0.3650 of the loss:

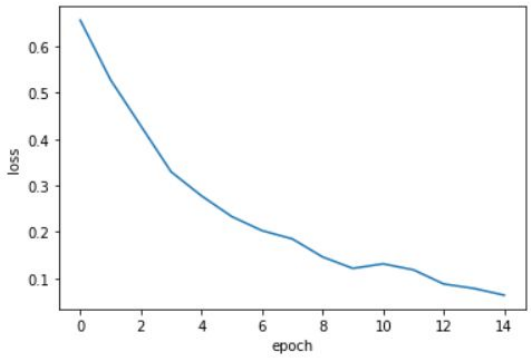
```
Epoch 15/15
1551/1551 [==============================] - 0s 51us/step - loss: 0.3650 - accuracy: 0.8446 - val_loss: 0.3777 - val_accurac
y: 0.8247
```

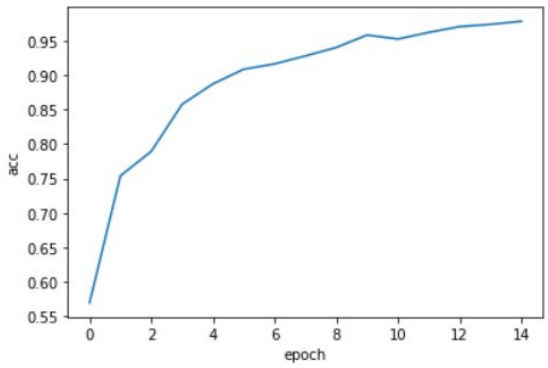




For CNN, after 15 echos, it can achieve 0.9781 of the accuracy and 0.0638 of the loss:

```
Epoch 15/15
1551/1551 [==============================] - 7s 5ms/step - loss: 0.0638 - accuracy: 0.9781 - val_loss: 0.2481 - val_accuracy:
0.9227
```

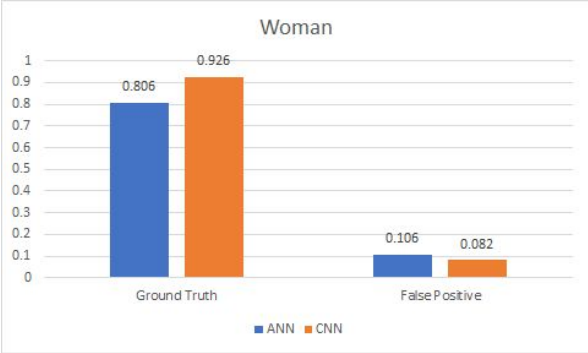


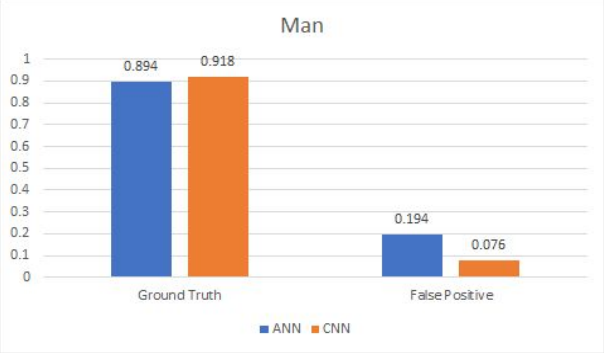


5.1.1.2 ANN vs CNN of Test predict performance

```
test dataset prediction result by ANN:
Man :
        Ground truth:  0.8941176470588236
        False positive:  0.19411764705882353
Woman :
        Ground truth:  0.8058823529411765
        False positive:  0.10588235294117647
```

```
test dataset prediction result by CNN:
Man :
        Ground truth:  0.9176470588235294
        False positive:  0.07647058823529412
Woman :
        Ground truth:  0.9235294117647059
        False positive:  0.08235294117647059
```



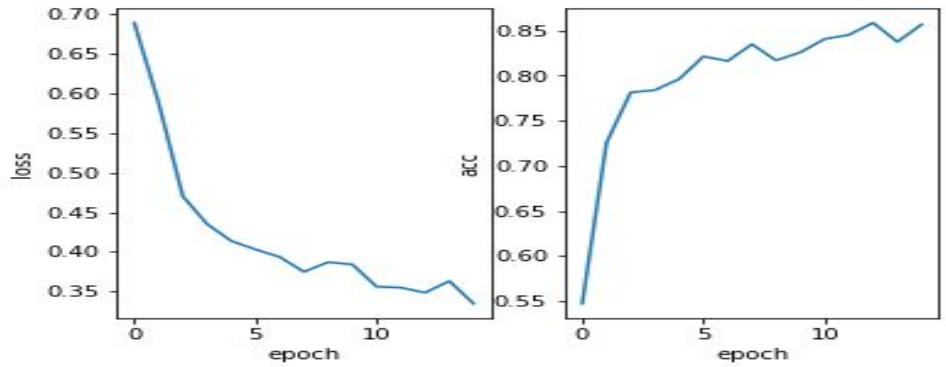


5.1.2 Try 100*100

5.1.2.1 ANN vs CNN of training performance:

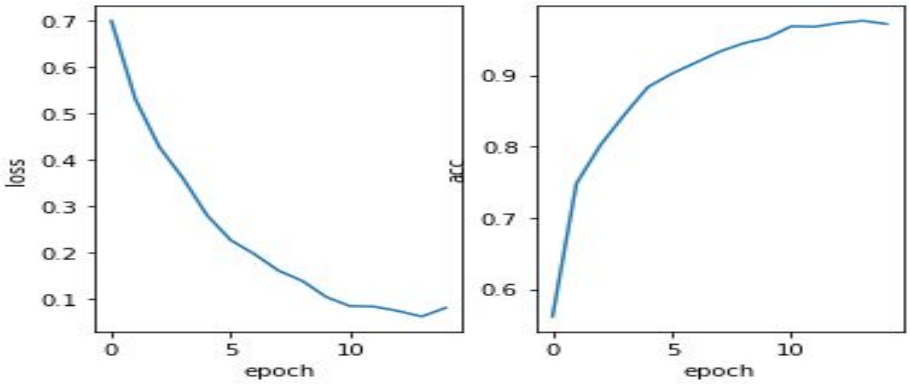
For ANN, after 15 echos, it can achieve 0.8569 of the accuracy and 0.3345 of the loss:

```
Epoch 15/15
1551/1551 [==============================] - 0s 174us/step - loss: 0.3345 - accuracy: 0.8569 - val_loss:
0.3600 - val_accuracy: 0.8557
```



For CNN, after 15 echos, it can achieve 0.9716 of the accuracy and 0.0818 of the loss:

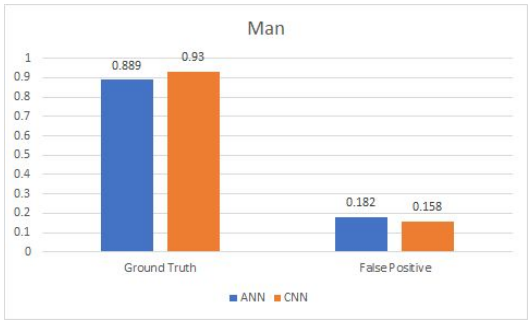
```
Epoch 15/15
1551/1551 [==============================] - 38s 25ms/step - loss: 0.0818 - accuracy: 0.9716 - val_loss:
0.4221 - val_accuracy: 0.8840
```

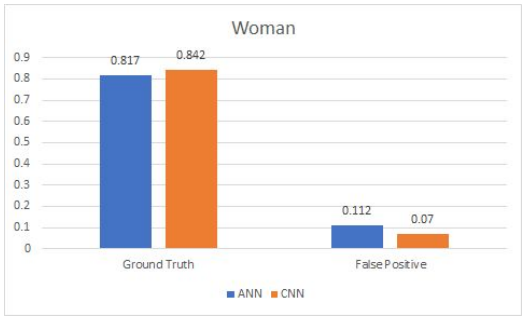


5.1.2.2 ANN vs CNN of test predict performance

```
test dataset prediction result by ANN:
Man :
        Ground truth:  0.888235294117647
        False positive:  0.18235294117647058
Woman :
        Ground truth:  0.8176470588235294
        False positive:  0.11176470588235295
```

```
test dataset prediction result by CNN:
Man :
        Ground truth:  0.9294117647058824
        False positive:  0.1588235294117647
Woman :
        Ground truth:  0.8411764705882353
        False positive:  0.07058823529411765
```



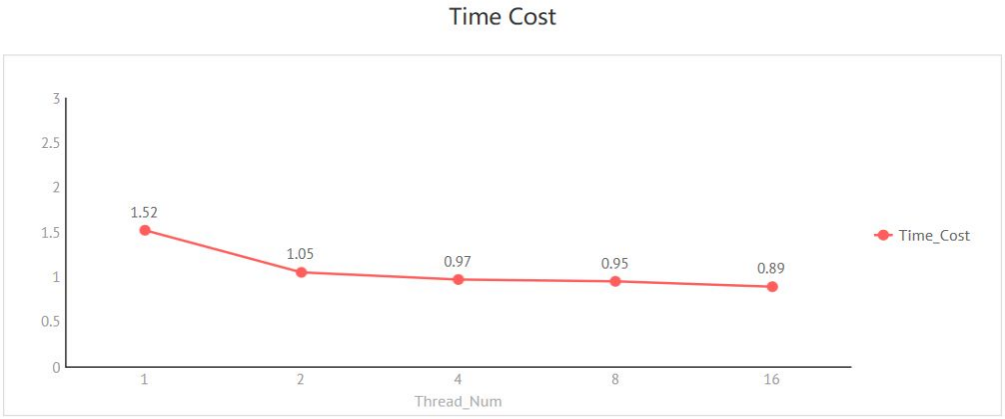


So, as can be seen from previous analysis, under the same size image, whether from the performance of the training set or the test set, CNN has better performance.

5.2 Performance of different Threads based on DNN model

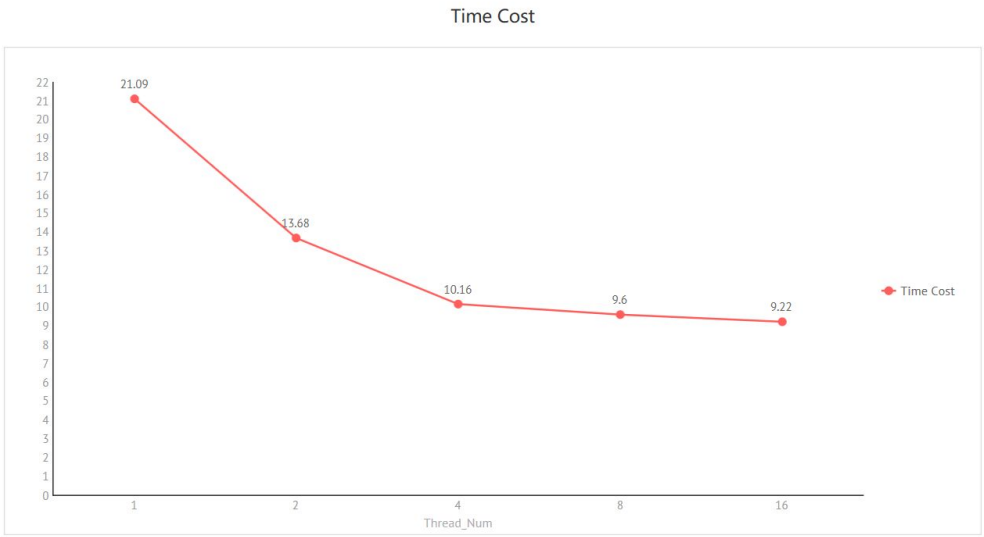
5.2.1 Change threads based on 28*28

Try different Thread_Num with 1,2,4,8,16, and we can see the result of speeding up as the following:



5.2.2 Change threads based on 100*100

By Increasing the size of the images, we can see a more obvious speed increase.



So, according to the above, the more threads are turned on, the faster the training speed and the higher the efficiency. But it is worth noting that it can open many threads blindly, which will slow down the calculation speed. In addition, if the size of the picture is too small, resulting in too short operation time, enabling parallel will cause the parallel overhead to be greater than the gain.

6.0 Conclusion

In this project, we mainly use the ANN and CNN for gender detection of the face image. We resize the image into different sizes to experiment and found that CNN has better performance on image classification task--CNN on Man's prediction precision is 0.918, ground truth is 0.076, on Woman's prediction precision is 0.924, ground truth is 0.082 on the setting of 28 * 28 size of the image data, which means a good performance. At the same time, we will process it in parallel to speed up the process. Moreover, within a reasonable range, the more input pixels (the larger the size), the more optimizations the parallel has.

According to the above analysis and results, we can conclude that in the field of image classification, convolution neural network performance(CNN) has better performance, whose accuracy can reach to 90% plus, and is more efficient than traditional machine learning algorithms and simple neural network algorithms. The layer structure of convolution neural network(CNN) , is like a combination of the feature engineering and classifier , so the convolution neural network (CNN) rather than extract the data feature in another process, the network will automatically extract the main features in the training of the network.

In future tasks, this model wouldn't be limited to gender detection (which is only a 2-category task), but would be considered for species classification (multi-category task) or speech recognition task and has a good performance based on the characteristics of convolutional neural network(CNN). In further experiments, exponential attenuation learning rate is also considered to optimize the network for better performance.

7.0 Reference

[1] G. Levi and T. Hassncer, "Age and gender classification using convolutional neural networks," 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Boston, MA, 2015, pp. 34-42.

Other link: https://keras.io/

8.0 Deliverables

1) Coding files：
    a) model_ANN_by Thread1.ipynb: use ANN model to detect gender by 1 thread
    b) model_CNN_by Thread1.ipynb: use CNN model to detect gender by 1 thread, compared with ANN
    c) my_ann_model.h5: saved trained ANN model
    d) my_cnn_model.h5: saved trained CNN model
    e) cnn_parallel-experiment.py: used to do parallel experiment and see the speed up
    f) my__cnn_thread_model.h5: saved model
2) Dataset
3) Final report
4) Final proposal
5) PPT Slides